# Piana High-Performance Data-as-a-Service (DaaS) Micro-service Framework

Yu Wang
College of Computing
Georgia Tech
Atlanta, USA

***Abstract*— The project "Piazzalytics" for the time being is, 1) to create a set of RestFul APIs to allow users extracting the entire online post information with the same hierarchical manner, 2) to persist the extracted data in the database and monitor the post communication, 3) to provide a functionality of notifying the users when new target posts/responses with desired keywords are created, and 4) finally to offer several analytical data services via the Restful APIs, including the historical records in the data warehouse and the data differential features. Although the current Piana data micro service project is still in its proof-of-concept stage, however, it has been already able to provide any further Piazza-related projects with the Piazza data service. Having said, this project could be even better if the JSON data from the Piazza could be further cleaned and reformatted to remove any irrelevant parts.**

***Keywords-data source, analysis, micro-serivce, DaaS.***

## I. INTRODUCTION

In this paper, I am going to describe the details of my development track project "Piazzalytics" as the final deliverable of the CS6460 course. Piazza, as a popular tool for the online interaction between instructors and students, provides a straightforward and user-friendly interface that is quite similar to an online forum. Students are able to put forward with various ideas, questions, and comments on the forum. In the meantime, the peer students and instructors are capable of responding to these topics and questions. From the perspective of common users, what the Piazza is currently offering is decent enough for constructing an online community. However, if an advanced user wishes to get all data from the Piazza in the format of programming-friendly data, the status quo is that the Piazza does not provide any data APIs for data reading or even data writing (e.g. creating a new post with a topic). Therefore, the project "Piazzalytics" for the time being is, 1) to create a set of RestFul APIs to allow users extracting the entire online post information with the same hierarchical manner, 2) to persist the extracted data in the database and monitor the post communication, 3) to provide a functionality of notifying the users when new target posts/responses with desired keywords are created, and 4) finally to offer several analytical data services via the Restful APIs, including the historical records in the data warehouse and the data differential features.

The contribution of this project could be significant. The "Piazzalytics" can be used for further business analysis, e.g. education/cognitive science/psychology related research. This could also make the instructors easier to track the course progress of the whole course participants. Meanwhile, the students could be more easily embedded into the courses. A more important influence is that, the data interfaces that this project realize can be really helpful to further online forum-related artificial intelligent field research. For instance, by using the Restful APIs of the "Piazzalytics", we can make a real-time robot (e.g. Piazzabot) that plays a teaching assistant role for the online courses. This kind of robot can be a great achievement of the artificial intelligence research.

TABLE I. THE FUNCTIONS OF PIANA DATA MICRO-SERVICE.

| Type | Description | Note |
|---|---|---|
| Restful APIs | Obtain Piazza data on post information | By giving the desired course ID @ /raw/{xxx} |
| Daemon job | Obtain and store the Piazza post data with unique timestamp in the MongoDB data warehouse, get the difference compared to the last data record, notify the downstream of the changes, persist the changes as well | Running on the daemon threads on background, per 12 hours by default, configurable. |
| Restful APIs | Obtain the historical data as well as the timestamp lists from the data warehouse, get the difference information between the data at two arbitrary timestamp for further analytical process, especially beneficial to time-series analysis. | By giving the desired course ID @ /raw/{xxx} |

## II. IMPLEMENTATIONS

The essence of this API project part is a web content extraction and scrapping on the Piazza course page (e.g. https://piazza.com/class/irl6njkfwh06de). As illustrated in the Figure 1, the modules of this whole project includes the Piazza URL pre-checking, web scrapping core, data binding utilities, Java Spring MVC as the central hub, data warehousing (MongoDB database), and data analysis.

After receiving an entered Piazza post URL via RestFUL POST (implemented by a Spring @RestController), the application would first examine if the URL is accurate and if the html contents conform to the predefined template for the scrapping at the next step (as a Spring @Service). It may also need to take the path parameters of username and login password (as Spring @PathParam) for the Piazza user authentication to obtain the HTML contents.

The Piazza contains the following data structure below.

- /class/irl6njkfwh06de: the main framework of the Piazza including the course information and all of the information in the left-hand navigation bar (indexed by cid);
- /class/irl6njkfwh06de?cid={cidValue}: all of the post threads that belong to a topic id cid;

Therefore, to highly efficiently obtain the data from the Piazza, I created a PiazzaDataSource class, which is to centrally download and parse the data from Piazza. As a result, based on the rules above, the scrapping core can hierarchically extract all of the posts, authors, responses, and date values, and finally store these data in the format of JSON. Based on the JSON data obtained, data binding modules then bind the JSON data to the objects, and these objects are then persisted into the data warehousing (MongoDB) with Spring MongoTemplate.

As the central functionality core, Java Spring is responsible to all of the functionality implementation as well as the API routings. Accordingly, the business logic module of data analysis may also be injected into the Spring context, which could analytically compute the statistics of the post, user, feedbacks, etc. Eventually, both of the Piazza data and analytical results are sent back to the user via RestFUL APIs in the format of JSON as the @ResponseBody.

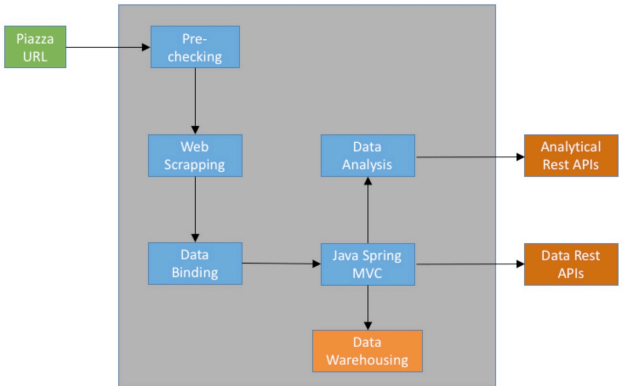


Figure 1. The conceptual scheme of the Piazza analytical API project.

TABLE II. MAJOR TECHNICAL SPECIFICATIONS

| Domain | Technology |
|---|---|
| Core framework | Spring Boot 1.4.1 |
| Language level | Java 1.8 |
| Build tool chain | Gradle 2.6 |
| Deployment | Docker 1.12.3 as the micro-service container |
| Database | MongoDB 3.2.7 |
| Message queue | JMS asynchronous message component |
| Cache | Guava Cache for database reading and network communication |
| Server | Apache Tomcat 8.5 server |
| API documentation | Swagger 2 |

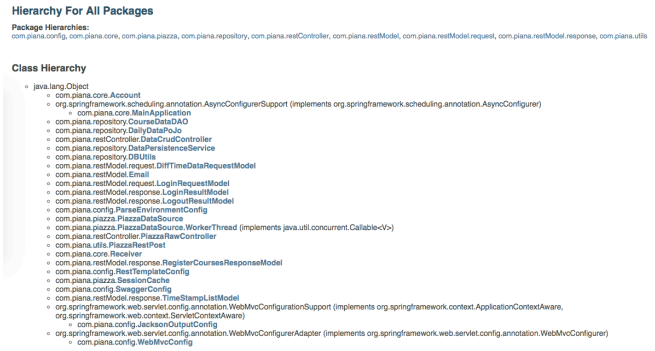


Figure 2. The hierarchical view on the Java classes.

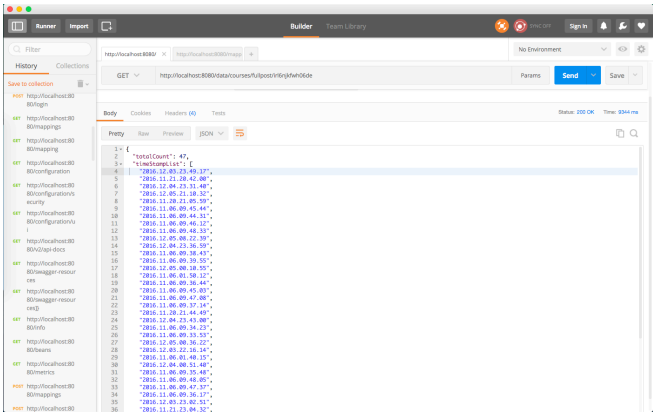

Figure 3. Example of time-series data warehousing records per timestamp.

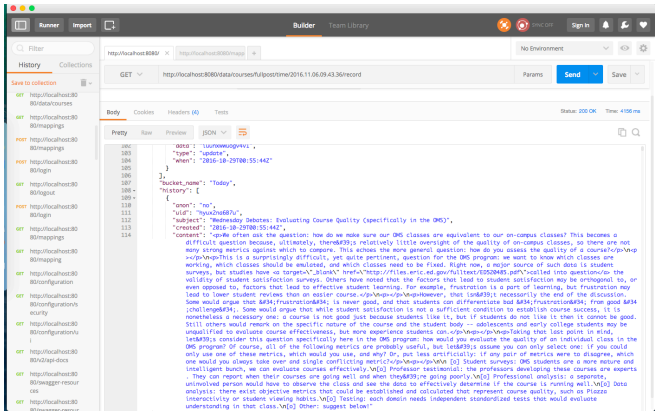

Figure 4. An example of JSON format data response from the data service.

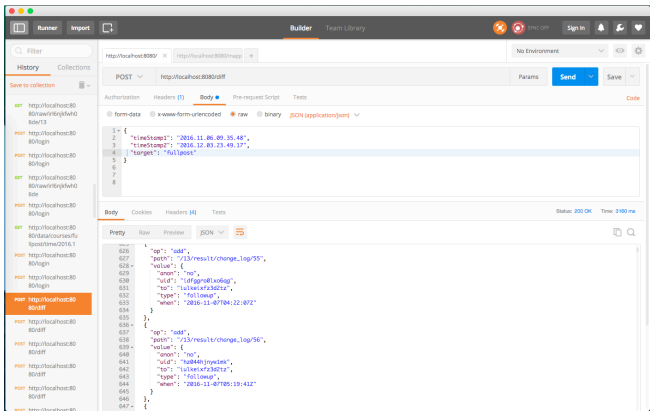

Figure 5. An example of data differentiation feature between the data at two timestamps.

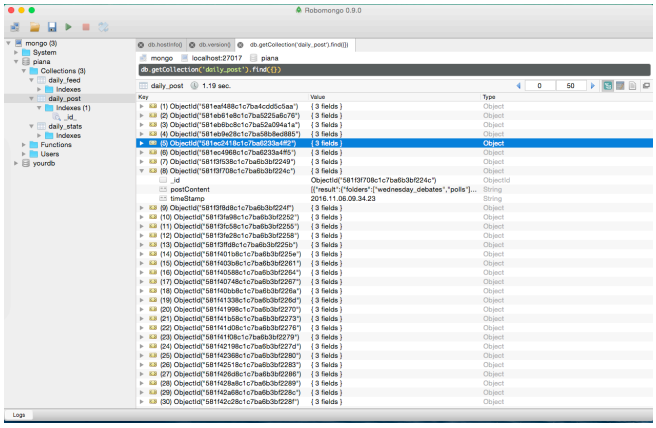

Figure 6. The screenshot of MongoDB database schemas and sample records.

## III. RESTFUL API LISTS

TABLE III. FULL LIST OF RESTFUL APIS

| Endpoint | Bean | Method |
|---|---|---|
| {[/data/courses],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public com.piana.restModel.response.RegisterCoursesResponseModel com.piana.restController.DataCrudController.registerCoursesForDatabasePersistence() throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/data/courses/feed],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public com.piana.restModel.response.TimeStampListModel com.piana.restController.DataCrudController.allTimeStampListofFeed() throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/data/courses/stats],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public com.piana.restModel.response.TimeStampListModel com.piana.restController.DataCrudController.allTimeStampListofStats() throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/data/courses/fullpost],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public com.piana.restModel.response.TimeStampListModel com.piana.restController.DataCrudController.allTimeStampListofAllPosts() throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/data/courses/feed/{nid}],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public com.piana.restModel.response.TimeStampListModel com.piana.restController.DataCrudController.allTimeStampListofFeedPerNid(java.lang.String) throws java.util.concurrent.ExecutionException,java. |

| | | |
|---|---|---|
| | | lang.InterruptedException |
| {[/data/courses/stats/{nid}],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public com.piana.restModel.response.TimeStampListModel com.piana.restController.DataCrudController.allTimeStampListofStatsPerNid(java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/data/courses/fullpost/{nid}],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public com.piana.restModel.response.TimeStampListModel com.piana.restController.DataCrudController.allTimeStampListofAllPostsPerNid(java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/data/courses/feed/time/{timestamp}/record],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.DataCrudController.feedContentPerTimeStamp(java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/data/courses/stats/time/{timestamp}/record],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.DataCrudController.statsContentPerTimeStamp(java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/data/courses/fullpost/time/{timestamp}/record],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.DataCrudController.postContentPerTimeStamp(java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/diff],methods=[POST],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.DataCrudController.getDifference(com.piana.restModel.request.DiffTimeDataRequestModel) throws |

| | | |
|---|---|---|
| | | java.util.concurrent.ExecutionException,java.lang.InterruptedException,java.io.IOException |
| {[/raw/{nid}/users],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.PiazzaRawController.getAllUsers(java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/raw/{nid}/stats],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.PiazzaRawController.getStatistics(java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/raw/{nid}/simplefeed],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.PiazzaRawController.getAllFeedSnippet(java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/login],methods=[POST],produces=[application/json]} | requestMappingHandlerMapping | public com.piana.restModel.response.LoginResultModel com.piana.restController.PiazzaRawController.login(com.piana.restModel.request.LoginRequestModel) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/logout],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public com.piana.restModel.response.LogoutResultModel com.piana.restController.PiazzaRawController.logout() throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/raw/{nid}/{cid}],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.PiazzaRawController.getContentPerCid(java.lang.String,java.lang.String) throws java.util.concurrent.Ex |

| | | |
|---|---|---|
| | | ecutionException,java.lang.InterruptedException |
| {[/raw/{nid}/userprofile],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.PiazzaRawController.getUserProfiles(java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/raw/{nid}/search],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.PiazzaRawController.getSearch(java.lang.String,java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/raw/{nid}],methods=[GET],produces=[application/json]} | requestMappingHandlerMapping | public java.lang.String com.piana.restController.PiazzaRawController.getAllPosts(java.lang.String) throws java.util.concurrent.ExecutionException,java.lang.InterruptedException |
| {[/v2/api-docs],methods=[GET],produces=[application/json application/hal+json]} | requestMappingHandlerMapping | public org.springframework.http.ResponseEntity<springfox.documentation.spring.web.json.Json> springfox.documentation.swagger2.web.Swagger2Controller.getDocumentation(java.lang.String,javax.servlet.http.HttpServletRequest) |
| {[/configuration/security]} | requestMappingHandlerMapping | org.springframework.http.ResponseEntity<springfox.documentation.swagger.web.SecurityConfiguration> springfox.documentation.swagger.web.ApiResourceController.securityConfiguration() |
| {[/configuration/ui]} | requestMappingHandlerMapping | org.springframework.http.ResponseEntity<springfox.documentation.swagger.web.UiConfiguration> springfox.documentation.swagger.web.ApiResourceController.uiConfiguration() |
| {[/swagger-resources]} | requestMappingHandlerMapping | org.springframework.http.ResponseEntity<java.util.List<springfox.documentation.swagger.web.SwaggerResource>> springfox.documentation.swagger.web.ApiResourceController.swaggerResources() |
| {[/error]} | requestMappingHandlerMapping | public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest) |
| {[/error],produces=[text/html]} | requestMappingHandlerMapping | public org.springframework.web.servlet.ModelAndView org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse) |
| {[/trace /trace.json],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke() |
| {[/info /info.json],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke() |
| {[/configprops /configprops.json],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke() |
| {[/actuator /actuator.json],produces=[application/json]} | endpointHandlerMapping | public org.springframework.hateoas.ResourceSupport org.springframework.boot.actuate.endpoint.mvc.HalJsonMvcEndpoint.links() |
| {[/env/{name:.*}],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EnvironmentMvcEndpoint.value(java.lang.String) |
| {[/env /env.json],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke() |

| | | |
|---|---|---|
| {[/heapdump /heapdump.json],methods=[GET],produces=[application/octet-stream]} | endpointHandlerMapping | public void org.springframework.boot.actuate.endpoint.mvc.HeapdumpMvcEndpoint.invoke(boolean,javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse) throws java.io.IOException,javax.servlet.ServletException |
| {[/beans /beans.json],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke() |
| {[/autoconfig /autoconfig.json],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke() |
| {[/dump /dump.json],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke() |
| {[/health /health.json],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.HealthMvcEndpoint.invoke(java.security.Principal) |
| {[/mappings /mappings.json],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke() |
| {[/metrics/{name:.*}],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.MetricsMvcEndpoint.value(java.lang.String) |
| {[/metrics /metrics.json],methods=[GET],produces=[application/json]} | endpointHandlerMapping | public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke() |

## IV. HIGH PERFORMANCE CHARACTERISTICS

TABLE IV. HIGH PERFORMANCE DESIGN

| Domain | Design |
|---|---|
| Piazza network communication | Run on independent thread, return Future<> as the result, store in the cache after the first visiting (with 5 min of expiry time by default) |
| Database query | Run on independent thread, return Future<> as the result, store in the cache after the first visiting (with 5 min of expiry time by default) |
| Get the entire full post data from Piazza | Use thread pools to concurrently obtain the data from the Piazza, and also on independent threads |
| Daemon background threads | Use JMS to asynchronously separate the comparison part from concrete treatment on change implementation. |

## V. SUMMARY

To sum up, although the current Piana data micro service project is still in its proof-of-concept stage, however, it has been already able to provide any further Piazza-related projects with the Piazza data service. Having said, this project could be even better if the JSON data from the Piaaza could be further cleaned and reformatted to remove any irrelevant parts.

## VI. ACKNOWLEDGEMENT

The author would like to thank all of the help and guidance form the Instructors and TAs.

## VII. BIBLIOGRAPHIES

[1] Jsoup. https://jsoup.org/

[2] Jaunt. http://jaunt-api.com/

[3] Selenium web driver. http://www.seleniumhq.org/projects/webdriver/

[4] jARVEST-web. http://sing.ei.uvigo.es/jarvest/